CSE 100: HASH TABLES (1)

Announcements

- Midterm review:
  - Materials posted on slides and resources page (we will add to this) – Don't just do the practice midterm! Also re-do clicker questions, review quiz questions, Stepik questions, and write PA1 code on paper!
  - Covers through (basics of) Hash tables
  - Review in discussion next week
  - Open TA office hours from 6:30-8pm next Tuesday in CSE 4140

Goals for today

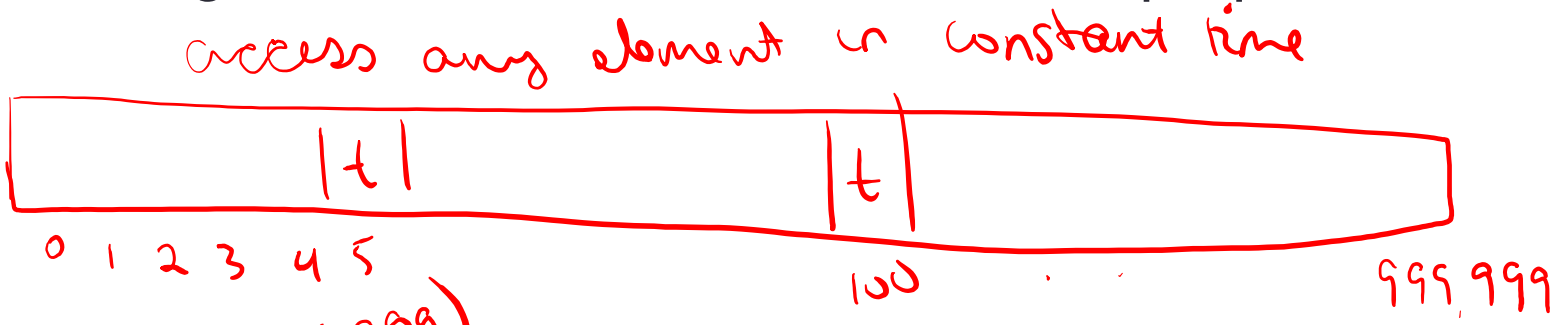
- Explain the idea behind and advantages of hashing and hash tables
- Calculate collision probabilities in hash tables

Finding data fast

Imagine that you want to store integers between 0 and 1,000,000. You want to be able to find out whether an element is present in your set. You know you can do this with a BST with average running time of O(N). But you decide to use an array with 1,000,000 Boolean values to store your data to try to make this faster. An entry will be true if the item is in your structure, and false otherwise.

What is the (Big-O) running time to "insert" an item into this proposed structure?

A. O(N)
B. O(logN)
C. O(1)

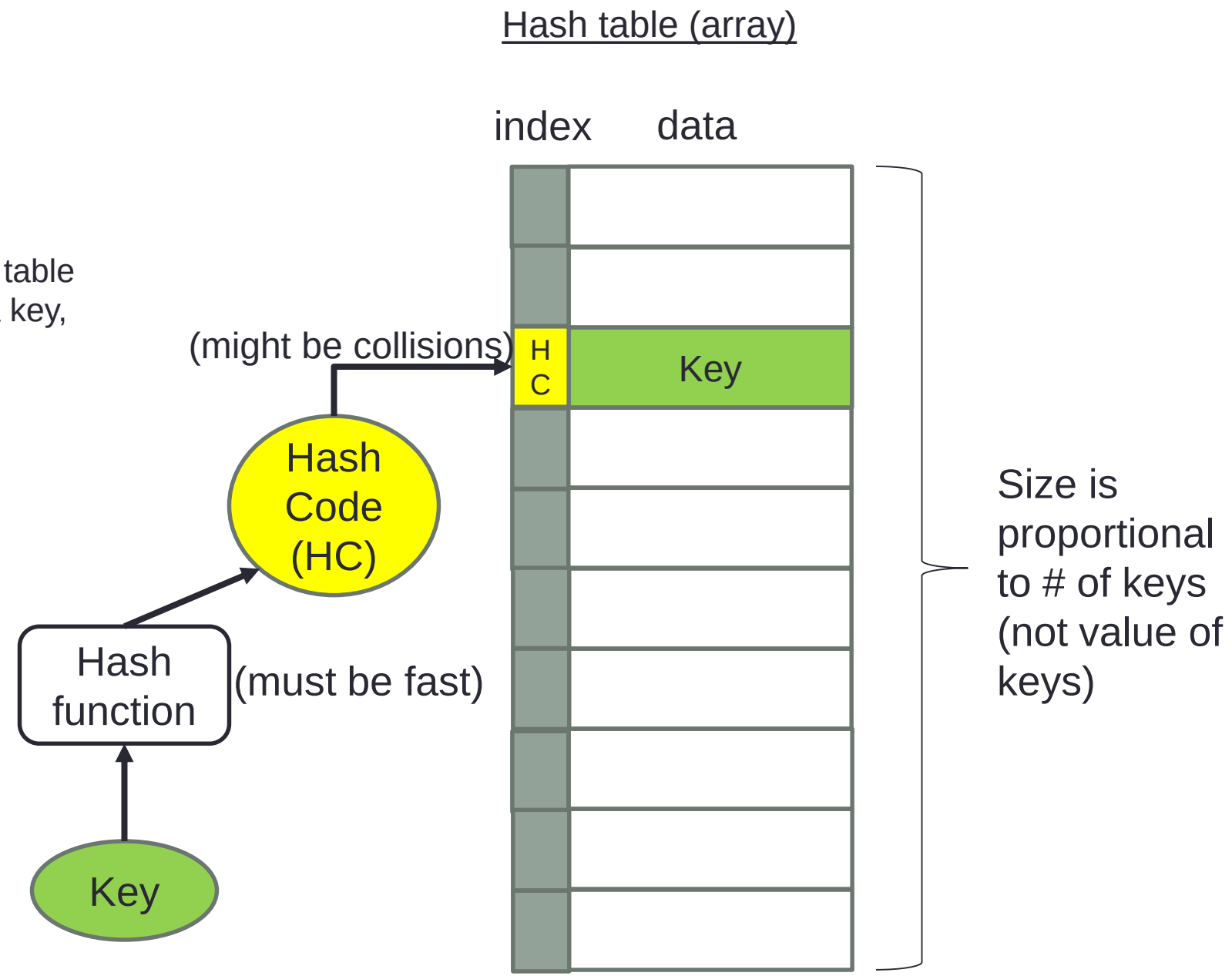


Problems
1) Limited in what I can store (ints, 0..999,999)
2) Very space inefficient

Hashing

- Let's modify our array-based look up table
- Need a hash-function h(x): takes in a key, returns an index in the array
- gold standard: random hash function

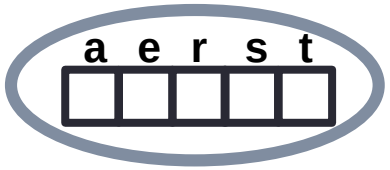
$h(k) = 42$



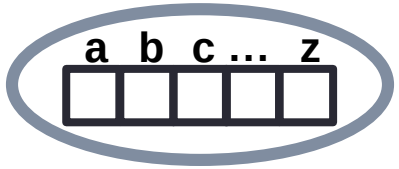
“Hashing” and MWTs

- The idea of hashing was at the heart of the MWT data structure we looked at. We mapped chars to ints to get their position in the array.

The hash function was:
H(‘a’) = 0
H(‘e’) = 1
H(‘r’) = 2
H(‘s’) = 3
H(‘t’) = 4



What is the hash function you would use for your PA if you implement a MWT with a vector? (ignore spaces for now)



H(c) = (int) c − (int) 'a'

This hash function has no collisions!

C++ container classes that implement hashtables

- unordered_set<class key>
- unordered_map<class key, class mapped_value>

Both of the above implement hashtables!

ADT (set)

hashset (unordered_set) | implemented with hashtable DS

```
unordered_set<int> hset;
for (int i=0; i<5; i++)
   hset.insert(i);
for (auto it = hset.begin();it!=hset.end();it++)
   cout<< *it << endl;
```

What will be printed by the above code?

A. The numbers 0 to 4 in sorted order
B. The numbers 0 to 4 in some arbitrary order
C. Neither A or B, we cannot use forward iterators with hash sets

hash function does not respect order

C++ container classes that implement hashtables

unordered_set<class key>
unordered_map<class key, class mapped_value>
Both of the above implement hashtables!

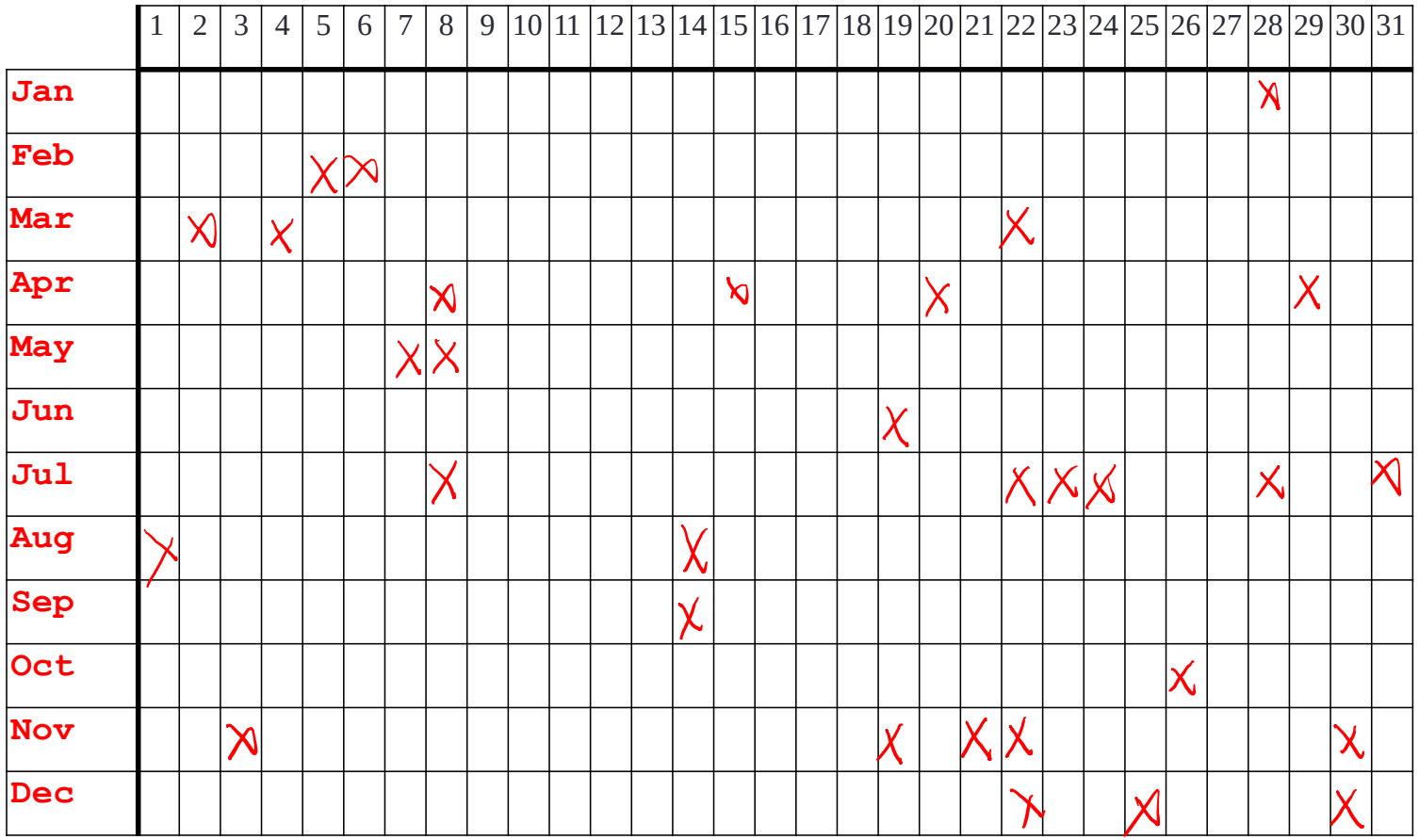
hash map map ADT
↓ implemented with
hash table DS

```
unordered_map<string, int> hmap;
vector<string> names{"aa", "bb", "cc", "dd", "ee"};
for (int i=0; i<5; i++)
   hmap[names[i]]=i;
for (auto it = hmap.begin();it!=hmap.end();it++)
   cout<< "("<<it->first << it->second<<") " ;
```

What will be printed by the above code?

A. (aa 0) (bb 1) (cc 2) (dd 3) (ee 4)
B. The above key value pairs but in arbitrary order
C. Neither A or B, we cannot use forward iterators with hash maps

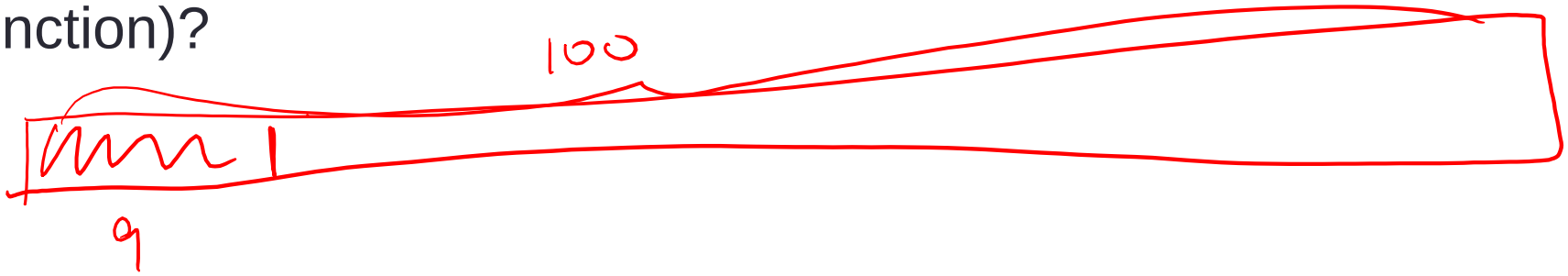
The birthday collision "paradox"



Probability of Collisions

- Suppose you have a hash table that can hold 100 elements. It currently stores 9 elements (in 9 different locations in the hash table). What is the probability that your next insert will cause a collision (assuming a totally random hash function)?

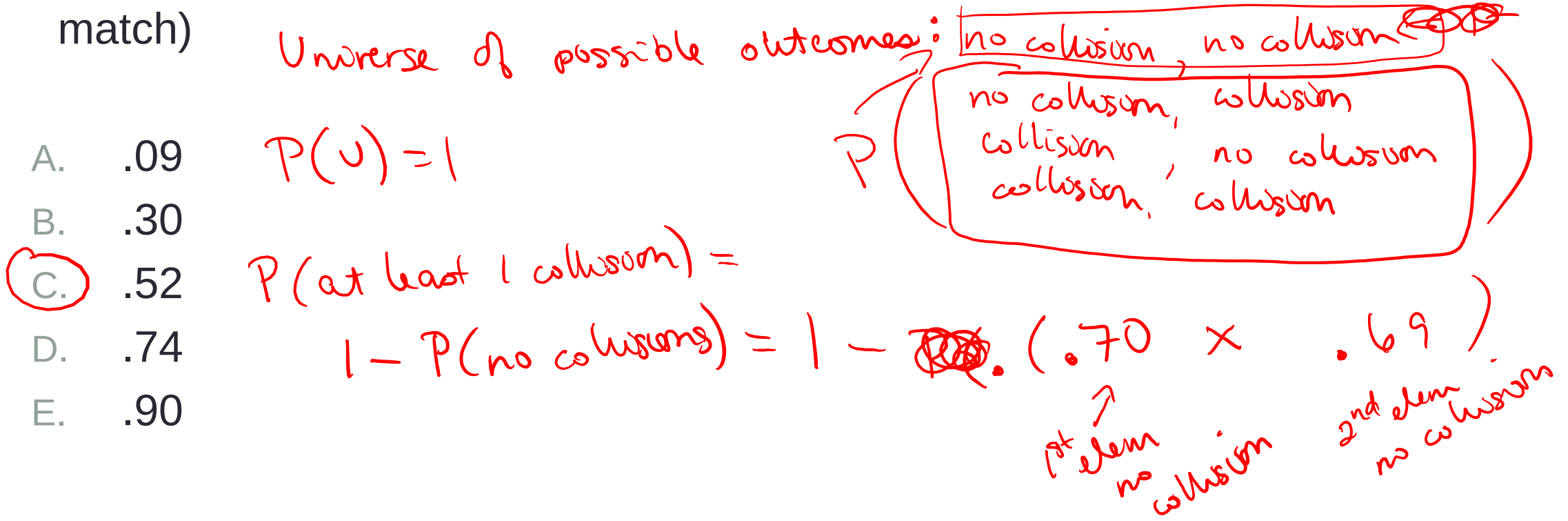
A. 0.09
B. 0.10
C. 0.37
D. 0.90
E. 1.00



Probability of Collisions

- Suppose you have a hash table that can hold 100 elements. It currently stores 30 elements (each in one of 30 possible different locations in the hash table). What is the probability that your next two inserts will cause at least one collision (assuming a totally random hash function)? (Choose the closest match)

A. .09
B. .30
C. .52
D. .74
E. .90



Probability of Collisions

- If you have a hash table with M slots and N keys to insert in it, then the probability of at least 1 collision is:

$$P_{N.M}(collision) = 1 - P_{N.M}(no\ collision)$$

$$= 1 - \prod_{i=1}^{N} P_{N.M}(ith\ key\ no\ collision)$$

Hashtable collisions and the "birthday paradox"

- Suppose there are 365 slots in the hash table: M=365
- What is the probability that there will be a collision when inserting N keys?
  - For N = 10, $prob_{N,M}$(collision) = 12%
  - For N = 20, $prob_{N,M}$(collision) = 41%
  - For N = 30, $prob_{N,M}$(collision) = 71%
  - For N = 40, $prob_{N,M}$(collision) = 89%
  - For N = 50, $prob_{N,M}$(collision) = 97%
  - For N = 60, $prob_{N,M}$(collision) = 99+%

- So, among 60 randomly selected people, it is almost certain that at least one pair of them have the same birthday
- In general: collisions are likely to happen, unless the hash table is quite sparsely filled
- So, if you want to use hashing, can't use perfect hashing because you don't know the keys in advance, and don't want to waste huge amounts of storage space, you have to have a strategy for dealing with collisions

PA2: Hash functions for strings

(Worksheet, problem 8)

- This hash function adds up the integer values of the chars in the string (*then need to take the result mod the size of the table*):

```
int hash(std::string const & key) {
  int hashVal = 0, len=key.length();
  for(int i=0; i<len; i++) {
    hashVal += key[i];
 }
 return hashVal;
}
```

Fast mean is constant in N

range of hash codes:

[8 * 'A' , 8 * 'z']

500 ~ 1000

- What is wrong with this hash function **for storing words of 8-characters**?

A. Nothing
B. It is too complex (takes too long) to compute ← maybe
C. It will lead to collisions between words with similar endings ← no
D. It will not distribute keys well in a large table ← definitely
E. It will never distribute keys well in any table ← maybe, depending on distribution in this range

PA2: Hash functions for strings

- On PA2 you will explore different hash functions for strings
- You can choose any two hash functions to compare
- You do NOT need to implement a full hash table
- More details in discussion *after* the midterm, but if you want to get started sooner, you can ask a tutor or TA if you have doubts.

Collision resolution strategies

- Unless weare doing "perfect hashing" we have to have a collision resolution strategy, to deal with collisions in the table.

- The strategy has to permit find, insert, and delete operations that work correctly!

- Collision resolution strategies you need to know are:

  - Separate chaining (from your reading)

  - Linear probing (from your reading)

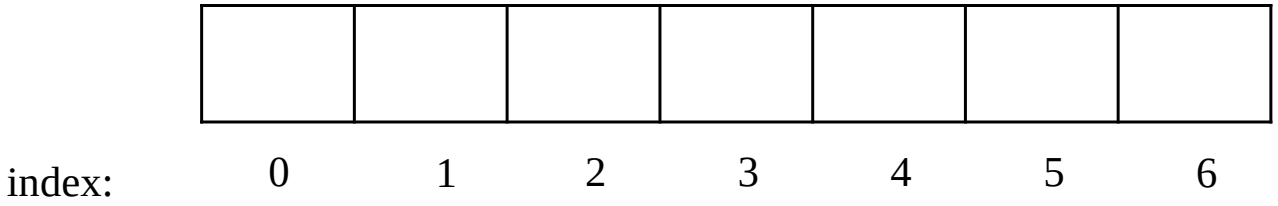
  - Double hashing (from your reading)

Resolving Collisions: Separate Chaining

- using the hash function H(K) = K mod M, insert these integer keys:

  701 (1), 145 (5), 218 (1), 12 (5), 750 (1)

  in this table:



Is there an upper bound to the number of elements that we can insert into a hash table of size M with separate chaining?

A. Yes, because of space constraints in the array
B. No, but inserting too many elements can affect the run time of insert
C. No, but inserting too many elements can affect the run time of find
D. Both B and C

What is the worst-case running time for insert and find for separate chaining?

Linear probing: inserting a key (Worksheet, problem 1 again)

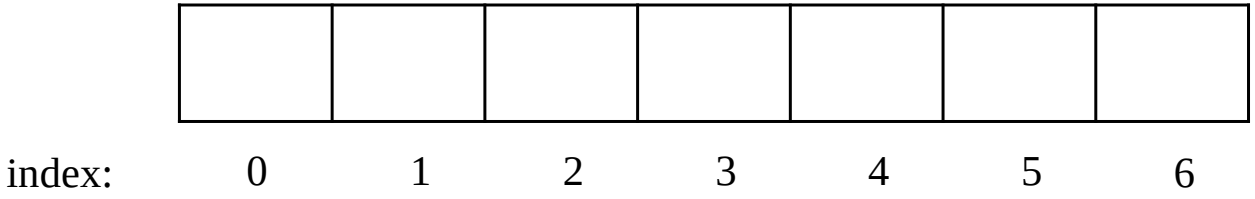
- When inserting a key K in a table of size M, with hash function H(K)

1. Set indx = H(K)
2. If table location indx already contains the key, no need to insert it. Done!
3. Else if table location indx is empty, insert key there. Done!
4. Else collision. Set indx = (indx + 1) mod M.
5. If indx == H(K), table is full! (Throw an exception, or enlarge table.) Else go to 2.

M = 7, H(K) = K mod M

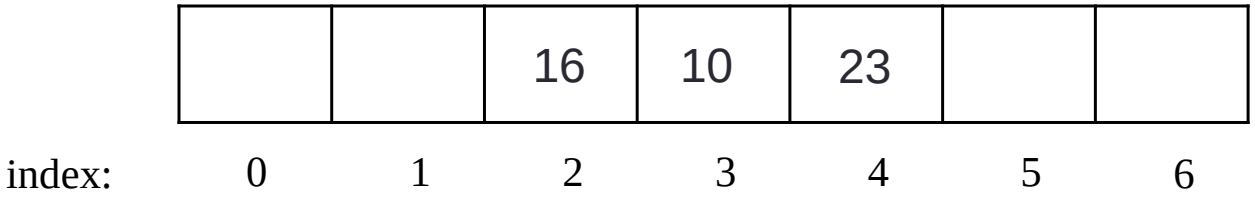
insert these keys 701 (1), 145 (5), 218 (1), 12 (5), 750 (1) in this table,

using linear probing:



Linear probing: searching for a key and deleting keys

- Using linear probing, can you delete an element by removing it completely from the table? Why or why not?
  A. Yes, it's fine to remove an element by removing it completely from the table.
  B. No, you cannot remove an element by removing it completely from the table.



find (23)
delete (10)
find (23)

Analysis of open-addressing (e.g. linear probing) hashing

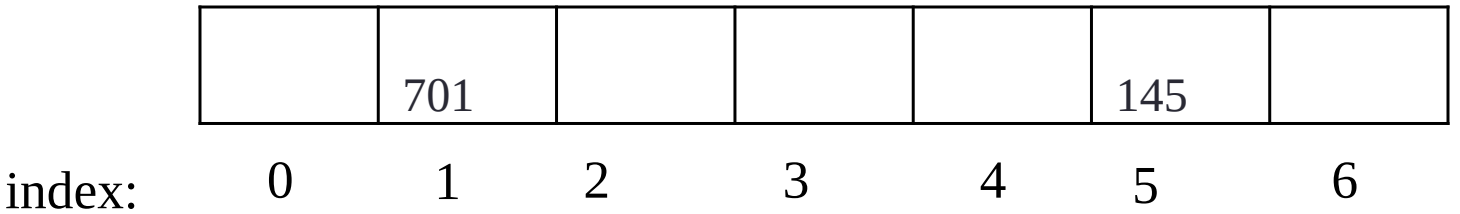
- What is the *worst-case* time to find a single element in a hash table with N elements in it?

A. O(1)
B. O(log(N))
C. O(N)
D. O(Nlog(N))
E. $O(N^2)$

Resolving Collisions: Double hashing

- A sequence of possible positions to insert an element are produced using two hash functions
- $h_1(x)$: to determine the position to insert in the array, $h_2(x)$: the offset from that position

701 (1,4), 145 (5,5), 218 (1,2), 12 (5,3), 750 (1,5)
in this table:



Hashtable worksheet (continued—do on your own)

A hash table of size 10 uses open addressing with hash function h(k) = k mod 10 and linear probing for collision resolution. After inserting 6 values into an empty hash table, the table looks like this:

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which of the following gives a possible order in which the key values could have been inserted into the table?

A. 46, 42, 34, 52, 23, 33
B. 34, 42, 23, 52, 33, 46
C. 46, 34, 42, 23, 52, 33
D. 42, 46, 33, 23, 34, 52

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Considering the same values and hash tables above, which of the following statements are true? (A=true, B=false)

1. 42 must have been inserted before 52
2. 34 must have been inserted before 52
3. 46 must have been inserted after 42
4. 33 must have been inserted after 46
5. 34 must have been inserted after 23

Now write down all of the other order constraints that must hold that you can think of.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

CHALLENGE: How many different insertion orders will lead to this hash table?

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

What is the probability of the next key that is inserted in the hash table from the previous problem going into each of the open spaces? Assume the value is chosen uniformly randomly (no value is more likely to be chosen than any other).

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

What is the probability that at least one of the next two keys inserted will cause a collision?

Consider a hash table of size 100. For what data will the hash function h(k) = k mod 100 definitely be a bad choice (select all that apply)?

1. If your keys are always greater than 100
2. If your keys are always multiples of 5
3. If your keys are always less than 50
4. If your keys are always multiples of 7

A. Definitely a bad choice: Will always distribute keys unevenly
B. Not necessarily a bad choice: Might distribute keys evenly